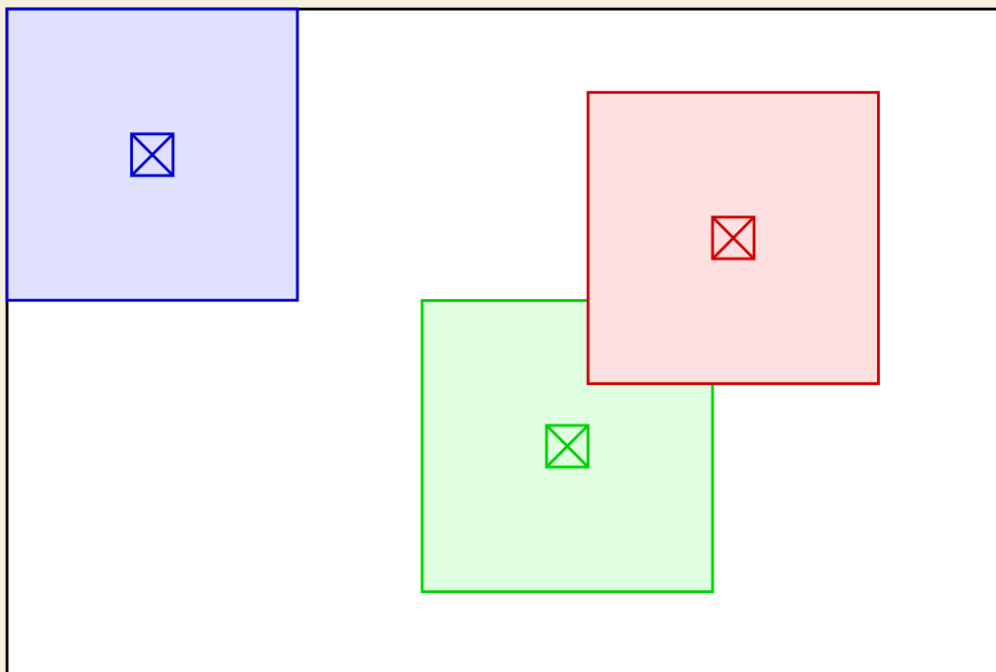


# **СИСТЕМЫ ТЕХНИЧЕСКОГО ЗРЕНИЯ**

БИНАРИЗАЦИЯ  
ИЗОБРАЖЕНИЙ.  
ФИЛЬТРАЦИЯ

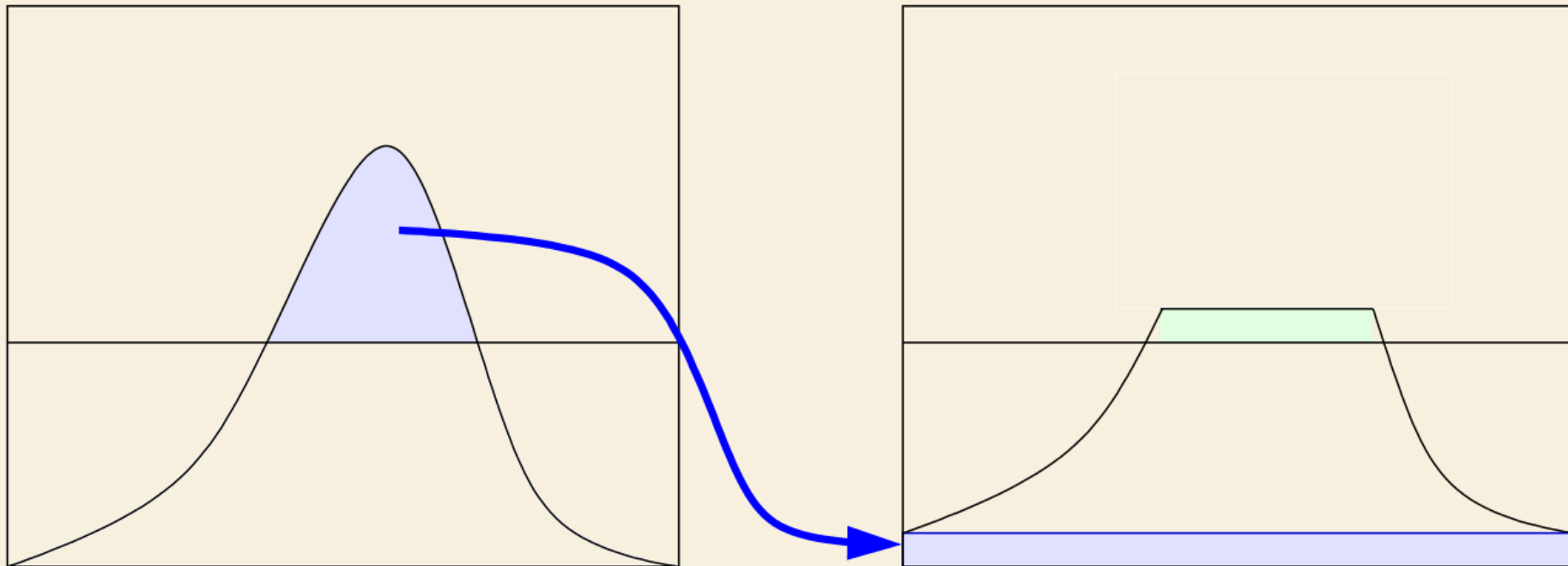
# АДАПТИВНОЕ ВЫРАВНИВАНИЕ



- Одноканальное изображение
- Мы будем вычислять новое значение яркости в пикселе, строя гистограмму по его окрестности
- При это среднюю яркость мы «подтягиваем» к середине диапазона
- *AHE* – Adaptive histogram equalization

# УЧЁТ КОНТРАСТА

- Мы можем обрезать выбросы на гистограмме
- Их можно учесть, сместив функцию распределения, «размазав» выброс по всей области
- *CLAHE* – Contrast limited AHE



```

#include <iostream>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgcodecs.hpp>

using namespace std;
using namespace cv;

int main()
{
    Mat src, dst;
    string fileName = "Picture1.png";
    src = imread(fileName, IMREAD_COLOR);
    if (src.empty())
        return -1;

    cvtColor(src, src, CV_BGR2HSV);
    Mat channels[3];
    split(src, channels);

    Ptr<CLAHE> pClahe = createCLAHE(80, Size(5,5));
    cout << pClahe->getTilesGridSize() << endl;
    cout << pClahe->getClipLimit() << endl;
    pClahe->apply(channels[2], channels[2]);
    merge(channels, 3, dst);

    cvtColor(dst, dst, CV_HSV2BGR);
    imshow("Result", dst);
    waitKey(0);

    return 0;
}

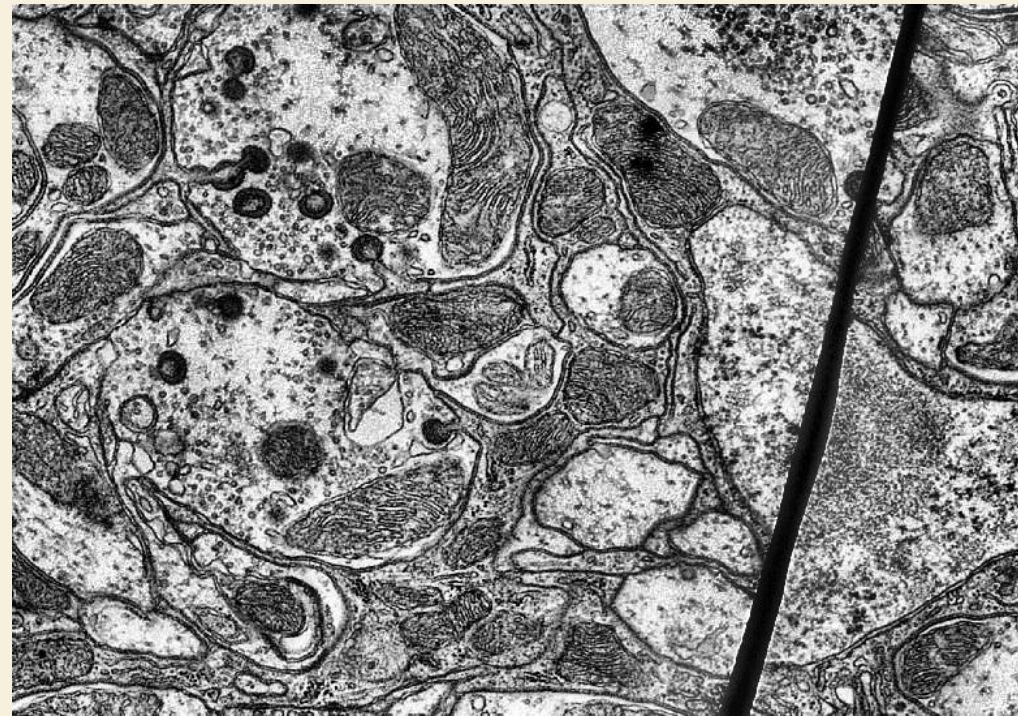
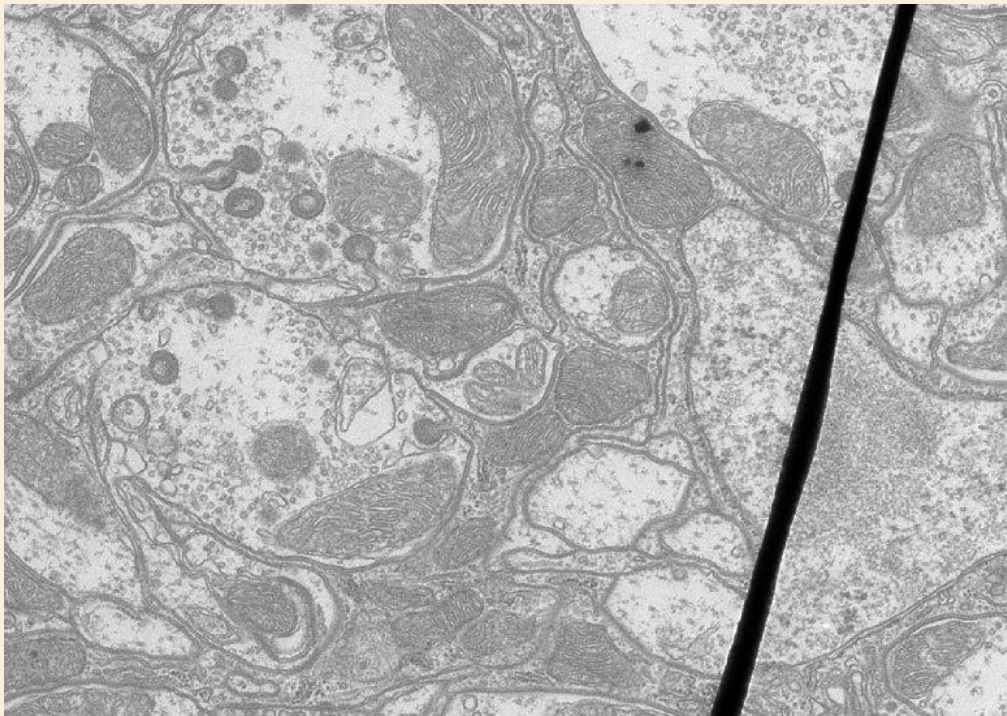
```

## В OPENCV

У метода createCLAHE 2 параметра:

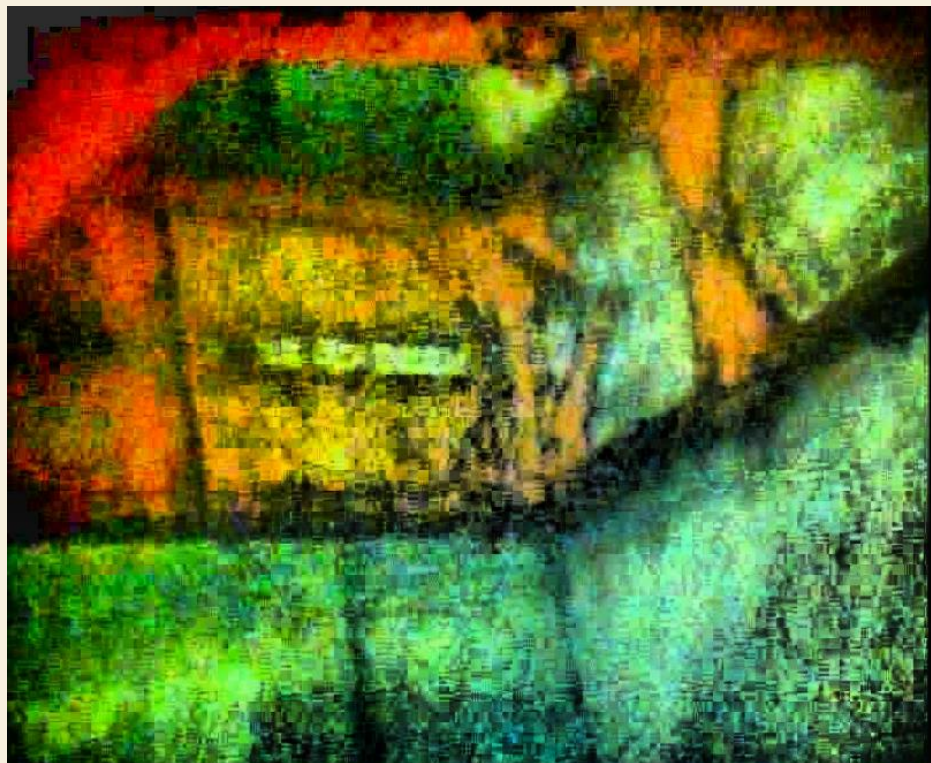
1. Ширина зоны до обрезки (по умолчанию 40)
2. Размер области, по которой вычисляется гистограмма (по умолчанию 8x8)

# РЕЗУЛЬТАТ





# И ЕЩЁ РЕЗУЛЬТАТ



# ШУМОПОДАВЛЕНИЕ

## **Соль и перец:**

случайные черные и  
белые пиксели

## • **Импульсный:**

случайные белые  
пиксели

## • **Гауссов:** колебания

яркости,

распределенные по  
нормальному закону



Original



Salt and pepper noise



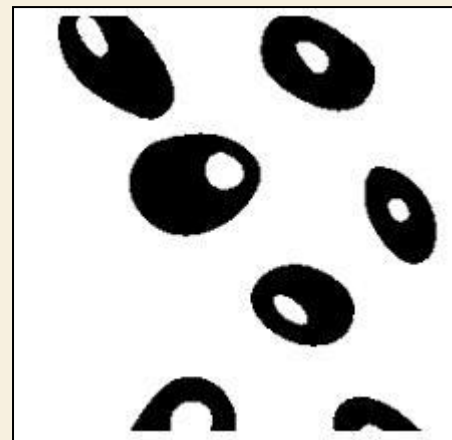
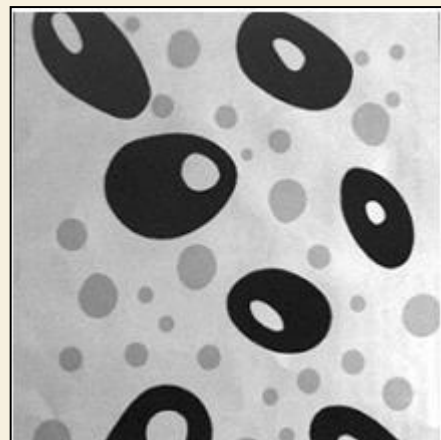
Impulse noise



Gaussian noise

# БИНАРНОЕ ИЗОБРАЖЕНИЕ

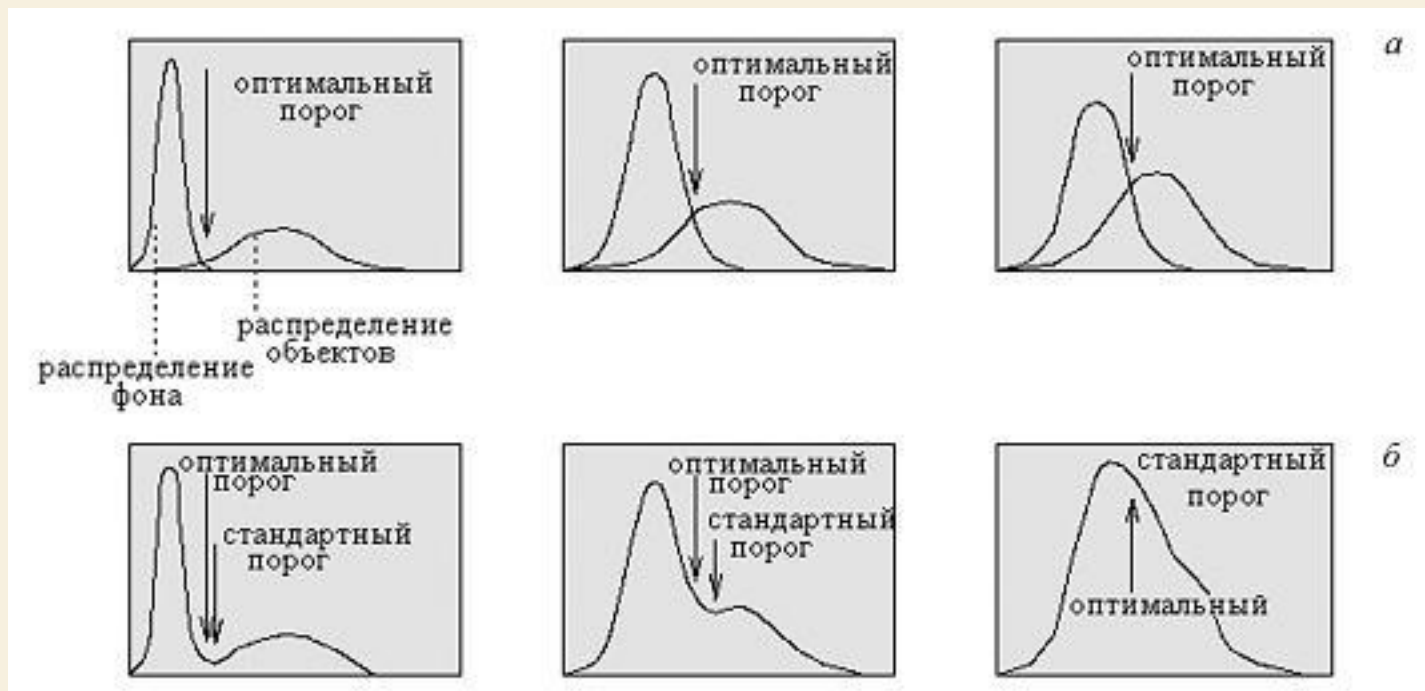
- Уменьшение объёма информации
- Упрощение её обработки
- Как её произвести?





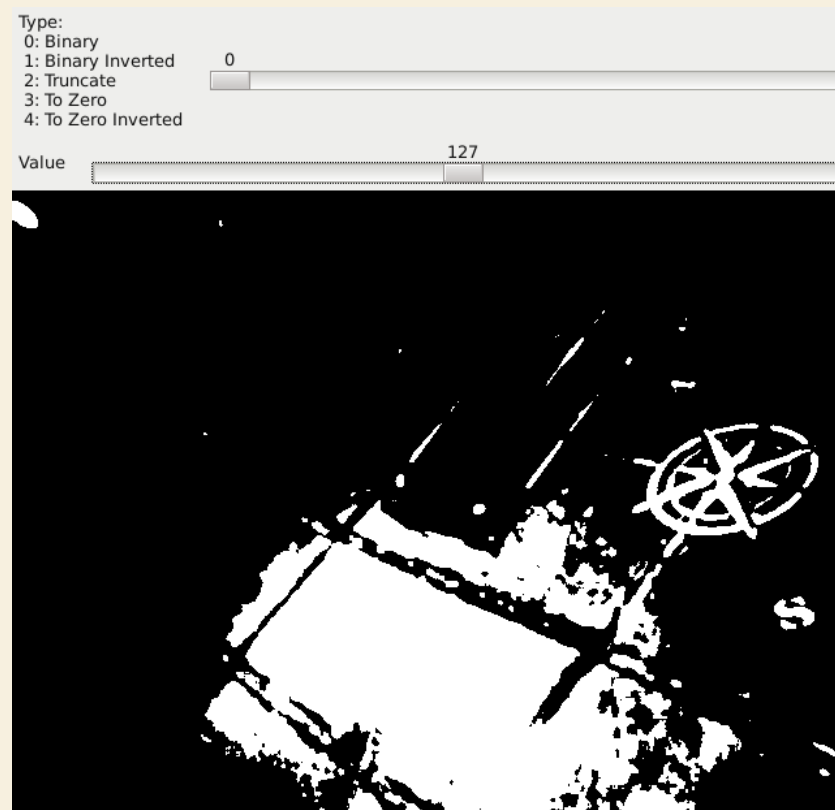
# БИНАРНОЕ ИЗОБРАЖЕНИЕ

- На основе обработки гистограммы яркостей
- Выбор конкретного положения?



# OPENCV\_BINARIZATION

ОРИГИНАЛ



# БИНАРИЗАЦИЯ

```
#include "opencv2/imgproc.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include <iostream>
using namespace cv;
using std::cout;
int threshold_value = 0;
int threshold_type = 3;
int const max_value = 255;
int const max_type = 4;
int const max_binary_value = 255;
Mat src, src_gray, dst;
const char* window_name = "Threshold Demo";
const char* trackbar_type = "Type: \n 0: Binary \n 1: Binary Inverted \n 2: Truncate \n 3: To Zero \n 4: To Zero Inverted";
const char* trackbar_value = "Value";
static void Threshold_Demo( int, void* )
{
    /* 0: Binary
       1: Binary Inverted
       2: Threshold Truncated
       3: Threshold to Zero
       4: Threshold to Zero Inverted
    */
    threshold( src_gray, dst, threshold_value, max_binary_value, threshold_type );
    imshow( window_name, dst );
}
```

```
int main( int argc, char** argv )
{
    String imageName("image.png"); // by default
    src = imread( imageName, IMREAD_COLOR ); // Load an image
    if (src.empty())
    {
        cout << "Cannot read the image: " << imageName << std::endl;
        return -1;
    }
    cvtColor( src, src_gray, COLOR_BGR2GRAY ); // Convert the image to Gray
    namedWindow( window_name, WINDOW_AUTOSIZE ); // Create a window to display results
    createTrackbar( trackbar_type,
                    window_name, &threshold_type,
                    max_type, Threshold_Demo ); // Create a Trackbar to choose type of Threshold
    createTrackbar( trackbar_value,
                    window_name, &threshold_value,
                    max_value, Threshold_Demo ); // Create a Trackbar to choose Threshold value
    Threshold_Demo( 0, 0 ); // Call the function to initialize
    waitKey();
    return 0;
}
```

MAIN()

# БИНАРНОЕ ИЗОБРАЖЕНИЕ

- Критерий Отсу (Оцу)
- Вычисляются математическое ожидание  $\mu$  и вероятность  $\omega$  для каждого значения яркости
- Вычисляются их суммы и ищется минимум

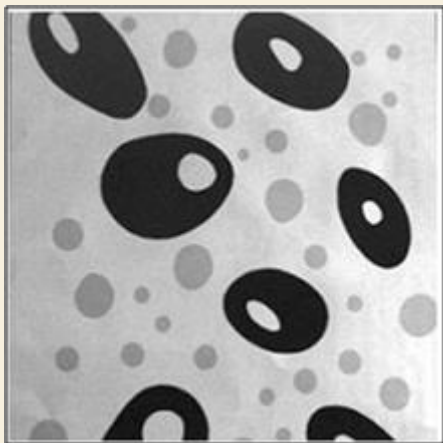
$$\sigma_b^2(t) = \sigma^2 - \sigma_\omega^2(t) = \omega_1(t)\omega_2(t)[\mu_1(t) - \mu_2(t)]^2$$

- Более подробное описание  
[https://docs.opencv.org/trunk/d7/d1b/group\\_\\_imgproc\\_\\_misc.html#gae8a4a146d1ca78c626a53577199e9c57](https://docs.opencv.org/trunk/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57)

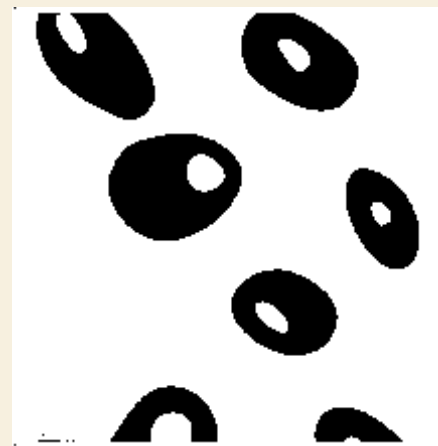


# БИНАРНОЕ ИЗОБРАЖЕНИЕ

ОРИГИНАЛ

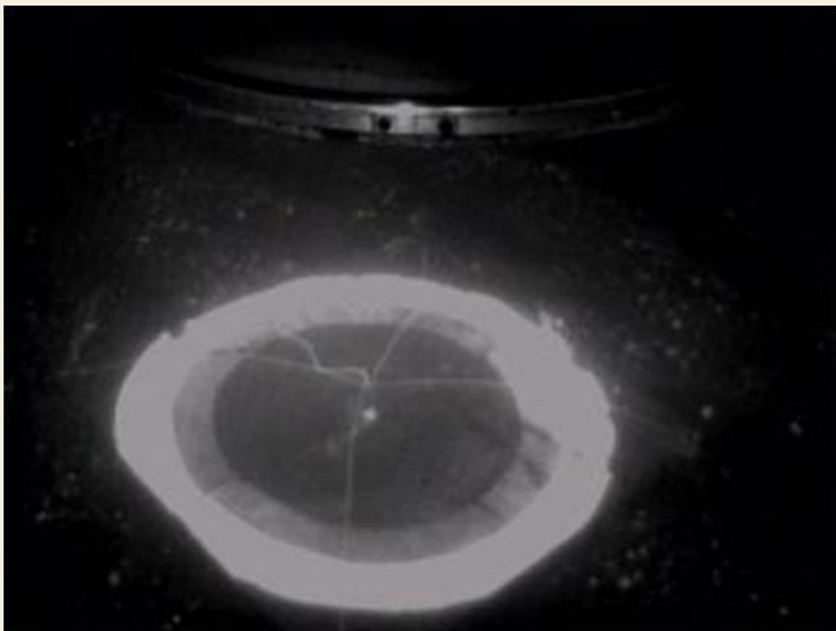


БИНАРИЗАЦИЯ THRESH\_BINARY



# БИНАРНОЕ ИЗОБРАЖЕНИЕ

ОРИГИНАЛ



БИНАРИЗАЦИЯ THRESH\_BINARY

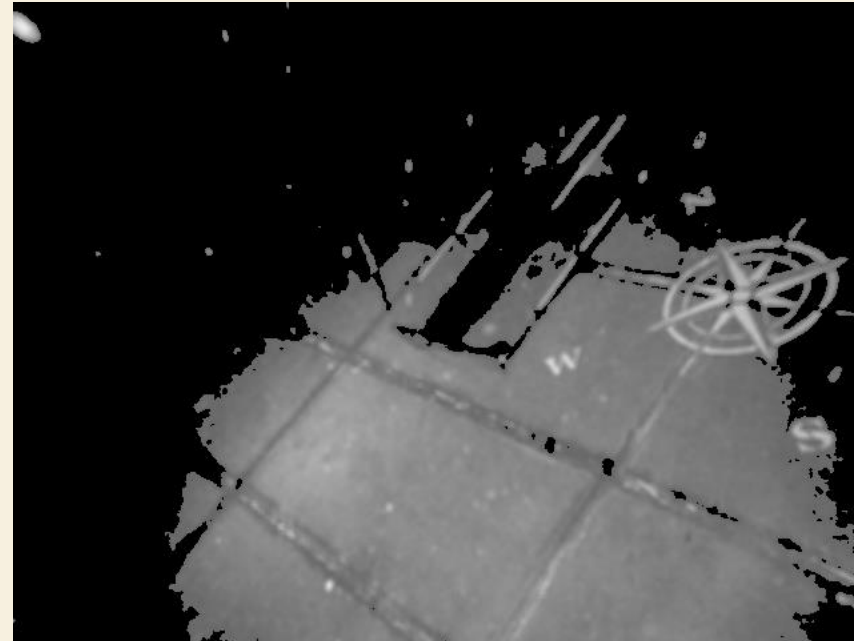


# БИНАРНОЕ ИЗОБРАЖЕНИЕ: А ЕСЛИ КонтРАСТНОСТЬ НИЖЕ?

ОРИГИНАЛ

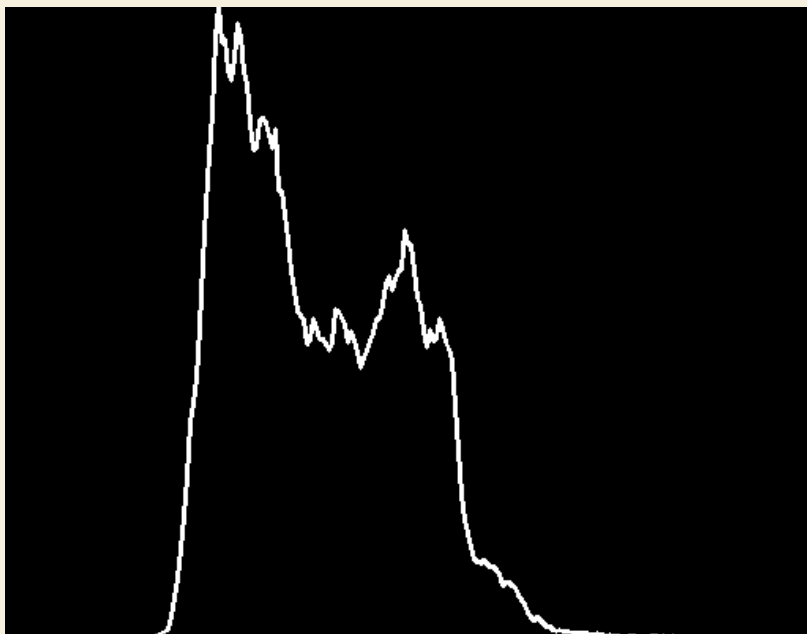


БИНАРИЗАЦИЯ TO\_ZERO

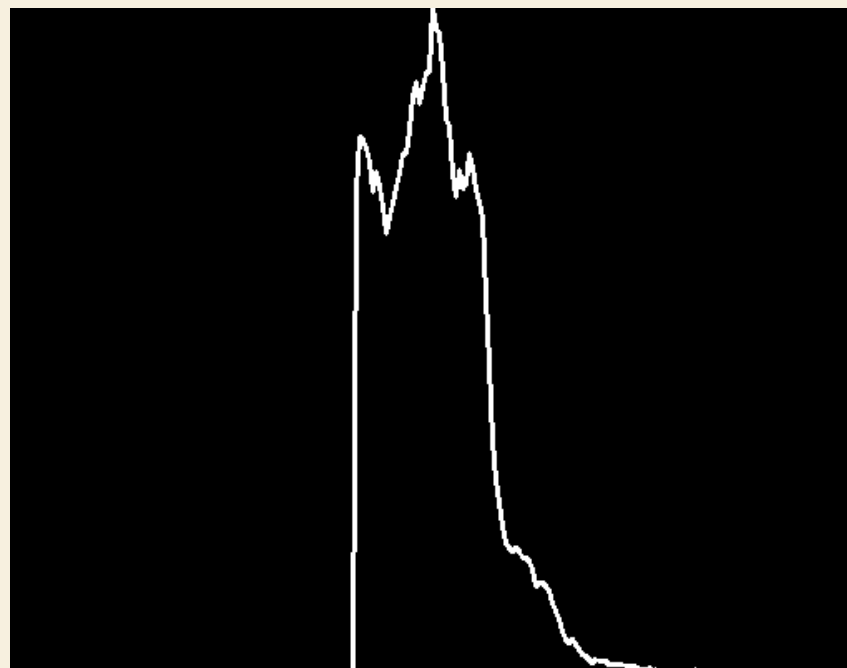


# БИНАРНОЕ ИЗОБРАЖЕНИЕ: А ЕСЛИ КонтРАСТНОСТЬ НИЖЕ?

ОРИГИНАЛ



БИНАРИЗАЦИЯ TO\_ZERO



# АДАПТИВНАЯ БИНАРИЗАЦИЯ

- Попробуем бинаризовать не всё изображение, а его отдельные области
- Точнее, будем сравнивать пиксель с его окружением
- Если он ярче порогового значения – 1, иначе – 0
- Как выбрать область?



# АДАПТИВНАЯ БИНАРИЗАЦИЯ

**MEAN 5X5**

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

**GAUSS 5X5**

1	2	3	2	1
2	4	6	4	2
3	6	9	6	3
2	4	6	4	2
1	2	3	2	1

# АДАПТИВНАЯ БИНАРИЗАЦИЯ

- Суммируем по области размером (size x size)
- При суммировании используем веса
- Делим на сумму весов

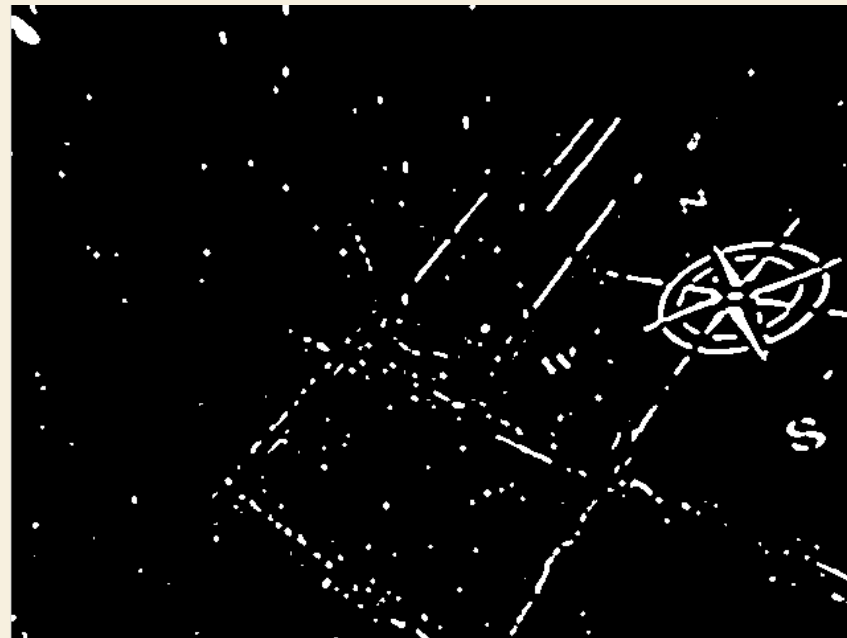
```
int cellSize = 12;          // Размер ячейки, n*n
int substrConst = 12;       // Значение, которое вычитается из среднего при бинаризации
adaptiveThreshold(src, dst, 255, ADAPTIVE_THRESH_GAUSSIAN_C, THRESH_BINARY, cellSize, 12);
```

# АДАПТИВНАЯ БИНАРИЗАЦИЯ

ОРИГИНАЛ

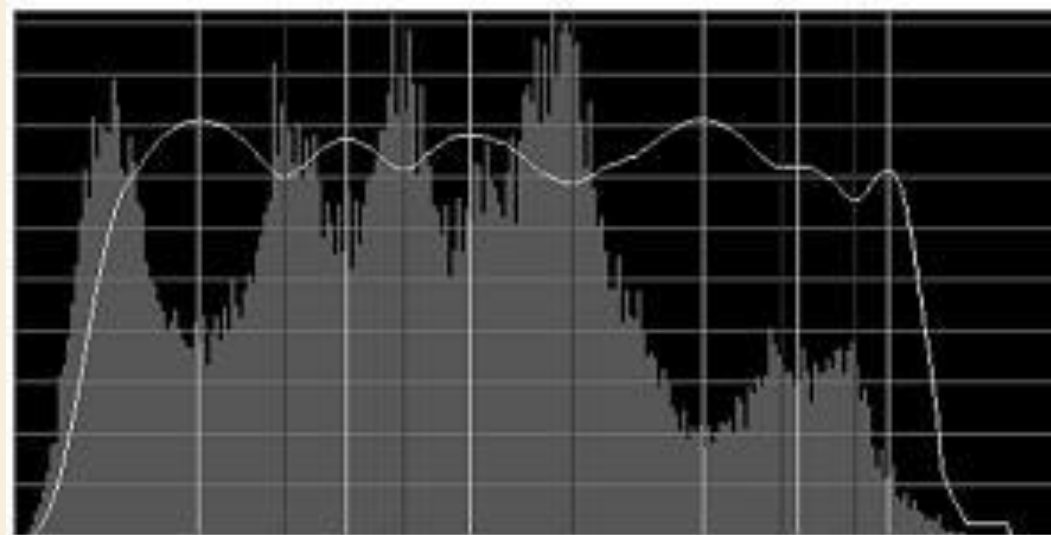


БИНАРИЗАЦИЯ MEAN



# СЕГМЕНТИРОВАННОЕ ИЗОБРАЖЕНИЕ

- Для многомодового случая нет универсального подхода

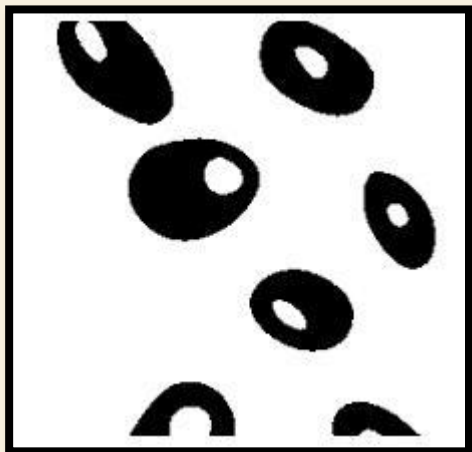




СЕГМЕНТИРОВАННОЕ  
ИЗОБРАЖЕНИЕ

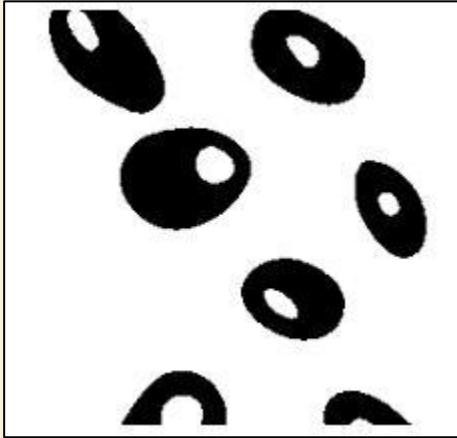


# БИНАРНОЕ ИЗОБРАЖЕНИЕ

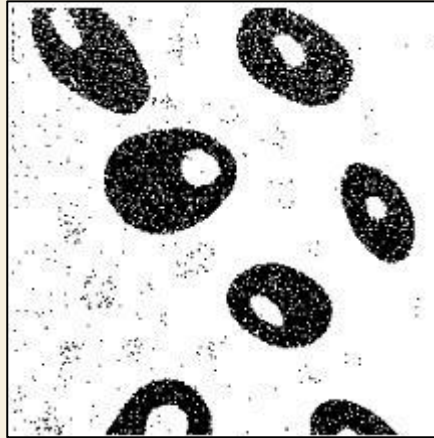


Модель шума «соль-перец»  
вероятности перехода

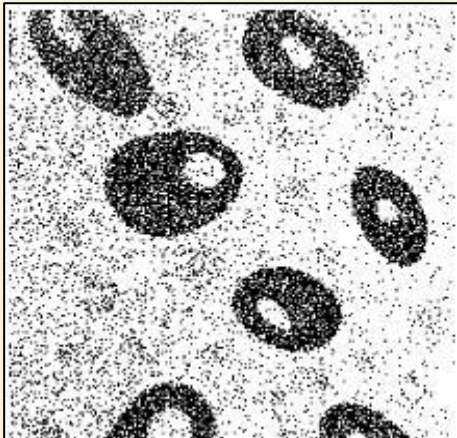
$Im[x,y] \rightarrow Im'[x,y]$	$Im'[x,y]=1$	$Im'[x,y]=0$
$Im[x,y]=1$	$1-p$	$p$
$Im[x,y]=0$	$q$	$1-q$



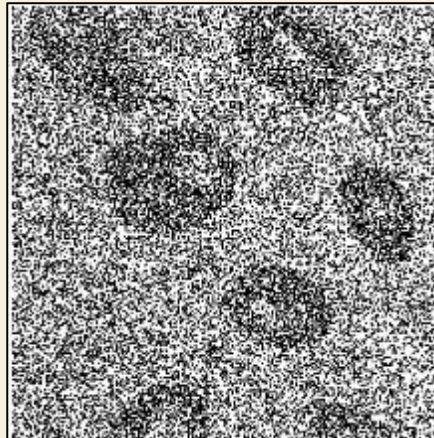
исходное



$p=0.1$   
 $q=0.1$



$p=0.25$   
 $q=0.25$



$p=0.45$   
 $q=0.45$

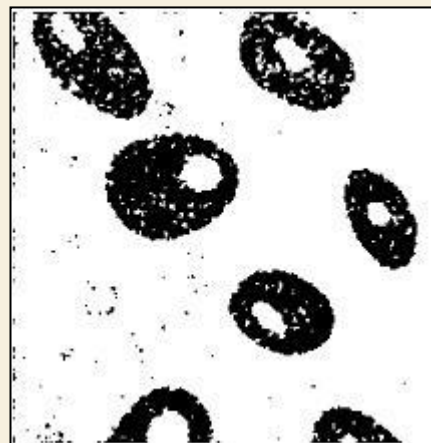
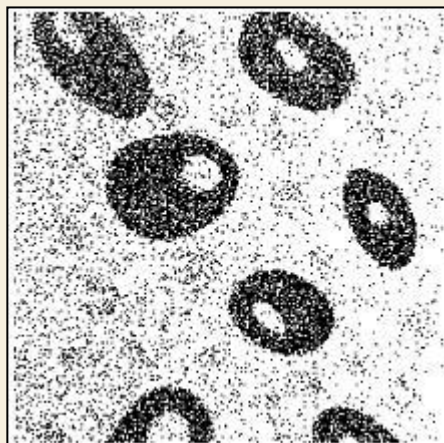
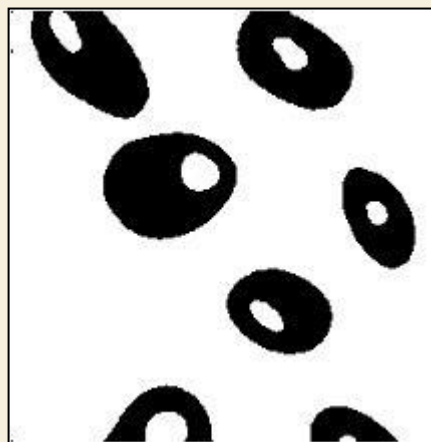
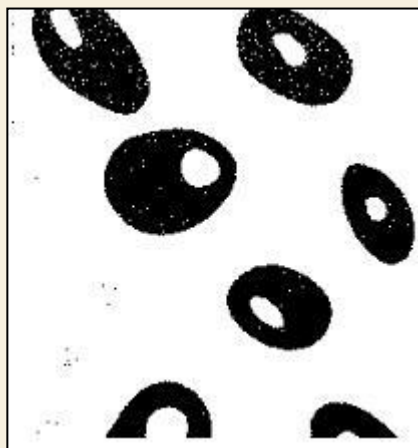
РАЗЛИЧНЫЙ  
УРОВЕНЬ ШУМА

# ФИЛЬТРАЦИЯ

- Медианный фильтр
- Считаем количество 0 и 1 в  $n$ -окрестности

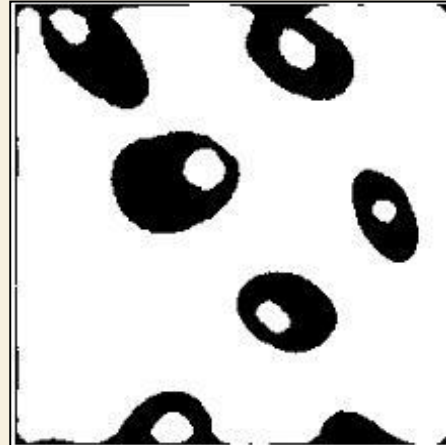
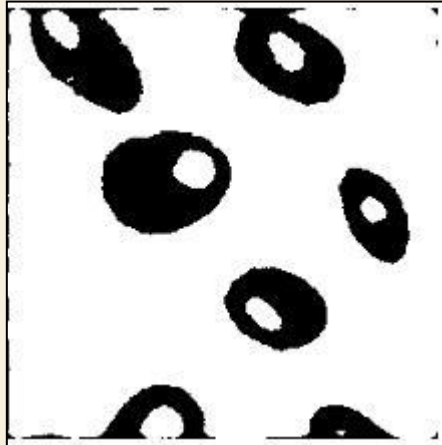
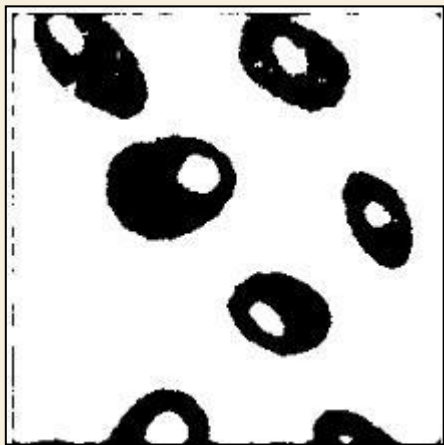
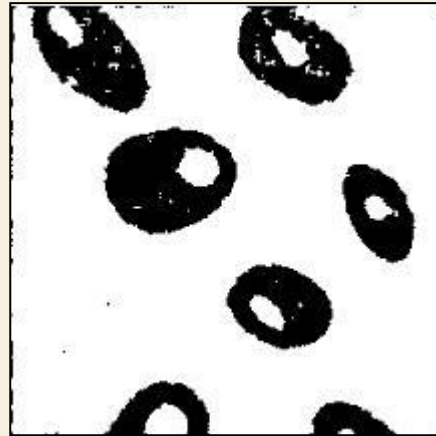
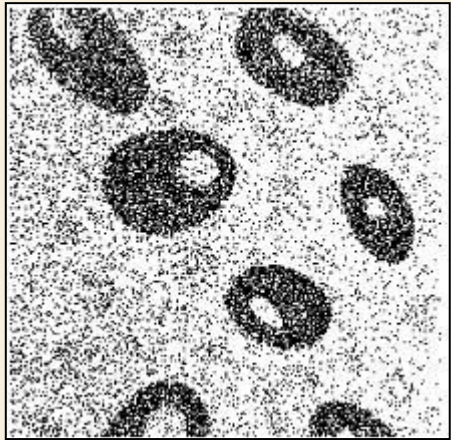
1	1	0
1	0	1
1	1	0

- Единиц > нулей  $\Rightarrow$  1



## БИНАРНОЕ ИЗОБРАЖЕНИЕ

Результат зависит от  
зашумлённости



## БИНАРНОЕ ИЗОБРАЖЕНИЕ

Результат зависит от  
размера апертуры – т.е.  
размера окна фильтрации



# РАНГОВЫЙ ФИЛЬТР

- ранговый фильтр – больше заданного порога
- разные пороги для нулей и единиц!
- Он же процентильный

1	1	0
1	0	1
1	1	0

- Единиц 6, нулей 3
- Ранг 6 – 1, ранг 7 – 0
- Нужен, если мы имеем априорную информацию